Demystifying GRPC

The Modern Toolkit for Building Microservices

Jason Wangsadinata
Systems Developer @ Brankas

First and foremost…

- Thank you to:
  - Python Indonesia for organizing this great event.
  - All the sponsors for making this event possible.
  - Kalbis Institute for hosting this event.

A little bit about myself

- **Jason Wangsadinata**
- `@jwangsadinata`
- Systems Developer @ Brankas
- Passionate about building scalable services and finding best practices to do so.
- Fun fact: I gave a talk on PyCon Indonesia last year too.

How do we make computers talk to each other?

How do we make *applications* that can talk to each other?

Web APIs

SOAP

REST

HTTP + JSON

(REST is not the above btw)

Why HTTP/REST?

- It is easy to understand (*text protocol*)
- Web infrastructure is already built on top of HTTP
- Great tooling for testing, inspection, modification
- Loose coupling between client/server makes changes relatively easy
- High-quality http implementations in every language

Why HTTP/REST - continued…

- **JSON**
  - It is simple
  - Looks like plain JavaScript object / Python dictionaries
  - XML was a little bit too verbose
- Movement away from SOAP(xml) towards REST

Looks like REST is perfect, bye…

Not quite…

- **JSON + HTTP is nice but it is not a silver bullet**
- It is not the best when:
  - **Performance matters**
  - Readability does not really matter
  - **Type safety is required**
  - Standard contract between applications/computers are needed (continued next page…)

Why REST API is not so good?

- **No formal (machine-readable) API contract**
  - Writing client libraries requires humans
  - **Humans (myself included) hate writing client libraries**
- Streaming is difficult
- Bi-directional streaming is not possible at all
- Inefficient (text representation are not efficient for networks)
- Hard to get many resources in a single request (think GraphQL)

“If I never write another REST client library in my life **I will die happy**”

- Alan Shreve, ngrok

Protocol Buffers

- Protocol buffers are a language-neutral, platform-neutral extensible mechanism for serializing structured data
- Google's Interface Description Language (IDL)
- Think XML, but smaller, faster, and simpler
- Has data types like `message`, `enum` and `service`.
- Language and platform neutral

“In our tests, it was demonstrated that this protocol performed up to **6 times faster** than JSON.”

https://auth0.com/blog/beating-json-performance-with-protobuf/

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;                                  -> set the syntax, either `proto2` or `proto3`
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;                                    -> set the package name

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {                          -> define the service
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}     -> define the method(s) in the service
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {                                -> now, define the message
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {                                -> now, define the message
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;                                   -> this message contains one field, `id`
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}                                          -> the fields (`int32`) are scalar types

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {                                   -> define another message as our rpc response
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;                    -> it has 3 fields, `name`, `id` and `rarity`
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;                          -> unlike `name` and `id`, `rarity` is an enum

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

-> Here’s how we define an enum in a protobuf

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;                    -> nit: unlike message, enum starts from 0
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;
}
```

I’m interested, how does it work?

- Define the message in a `.proto` file

```
syntax = “proto3”;
package pokemon;

service PokemonService {
    rpc Get (GetReq) returns (Pokemon) {}
}

message GetReq {
    int32 id = 1;
}

message Pokemon {
    string name = 1;
    int32 id = 2;

    enum Rarity {
        NORMAL = 0;
        LEGENDARY = 1;
        MYTHIC = 2;
    }

    Rarity rarity = 3;                          -> and here’s how to use it
}
```

What's next?

- The file is actually machine-readable, and can be read by a compiler
- The compiler can then generate the client libraries by the following:
- **Python**
  - `protoc -I=$SRC_DIR --python_out=$DST_DIR $SRC_DIR/pokemon.proto`
- Go
  - `protoc -I=$SRC_DIR --go_out=$DST_DIR $SRC_DIR/pokemon.proto`
- JavaScript
  - `protoc -I=$SRC_DIR --js_out=$DST_DIR --grpc_out=$DST_DIR --plugin=proto-gen-grpc=`which grpc_node_plugin` $SRC_DIR/pokemon.proto`
- etc

We have client libraries for the API in multiple languages

- Python
- Go
- C++
- Java (also works on Android)
- Ruby
- C#
- JavaScript (Node.js)
- Objective-C (hence, iOS)
- PHP

quick demo

yay!!

Finally, GRPC!

GRPC: What is it?

- A high performance, open-source, universal RPC framework
- It stands for **g**RPC **R**emote **P**rocedure **C**alls
- Part of Cloud Native Computing Foundation (cncf.io)
- Open-source version of *Stubby* used in *Google.*

GRPC — continued

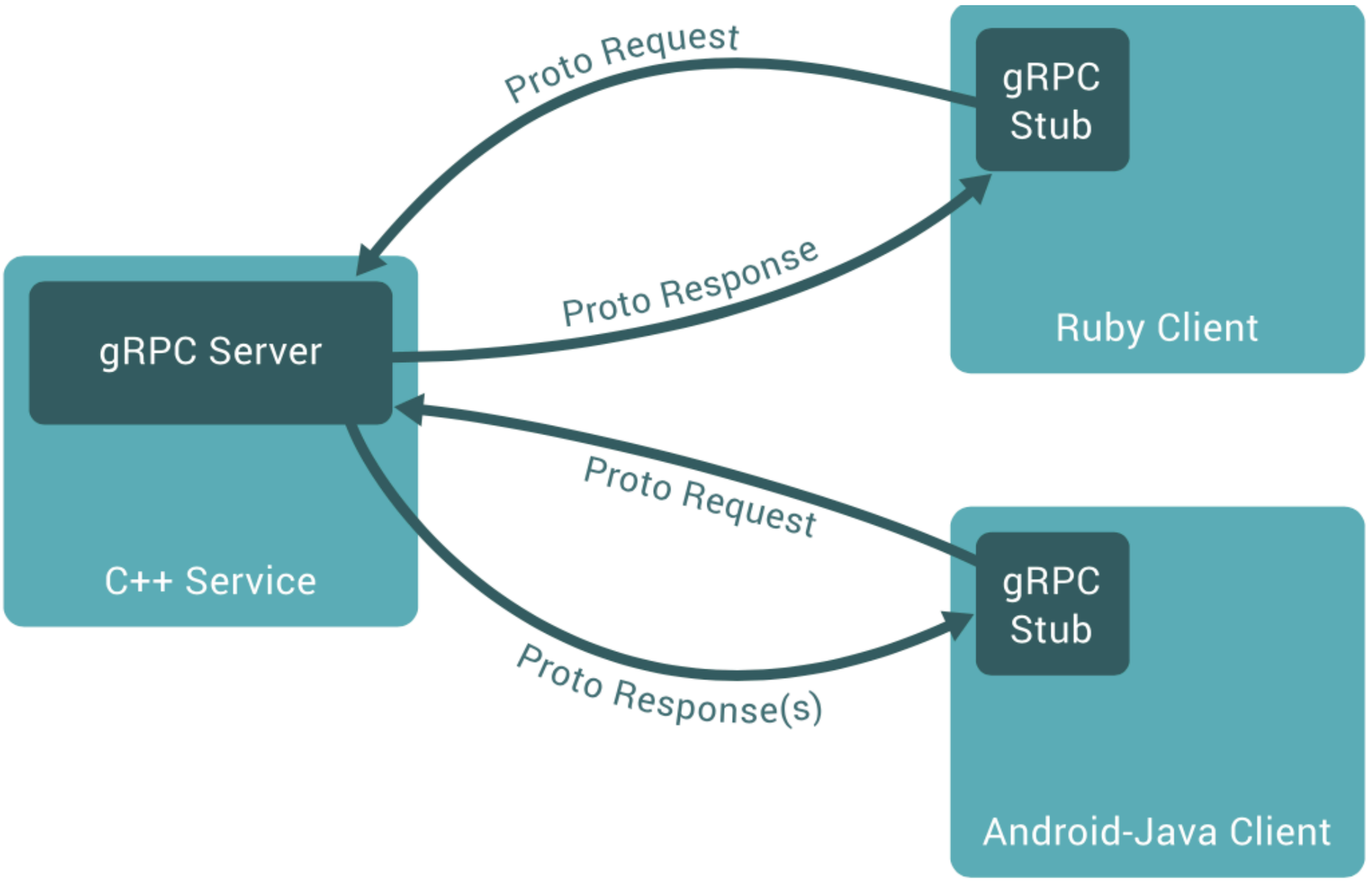
- Abstractions and best practices on how to design RPCs
- Default implementation(s) from Google
- Extension points to plug custom implementations and modifications
- Supports 10+ programming languages

- In particular, Python has first class support for *protobuf* and *gRPC*.

GRPC: On the wire

- HTTP/2
- `protobuf` serialization (pluggable)
- Clients open one long-lived connection to a GRPC server
  - A new HTTP/2 stream for each RPC call
    - Allows simultaneous in-flight RPC calls
- Allows client-side and server-side streaming

- Built on:
  - `HTTP/2, IDL, protobufs`


Proto Request
gRPC
Stub
Proto Response
gRPC Server
Ruby Client
Proto Request
gRPC
Stub
Proto Response(s)
C++ Service
Android-Java Client

GRPC: Implementation

- Three high-performance event loop driven implementations
- **C**
  - Ruby, Python, Node.js, PHP, Objective-C, C++, C# are all bindings to the `C Core`
  - PHP via PECL extension (apache or nginx/php-fpm)
- **Java**
  - Netty + BoringSSL via JNI
- **Go**
  - Pure Go implementation using Go stdlib `crypto/tls` package

enough theory, let’s code

Generate some Python code

- Similarly, we want to use the `protoc` compiler to generate the Python artifacts
- `$ python3 -m grpc_tools.protoc -I=. --python_out=./python --grpc_python_out=./python pokemon.proto`

- This will generate the following:
  - Classes for the messages on `pokemon.proto`
  - Classes for the services on `pokemon.proto`
    - `PokemonServiceStub, PokemonServiceServicer`
  - Function for the service defined on `pokemon.proto`
    - `add_PokemonServiceServicer_to_server`

Write the server

```
# imports are omitted for clarity

class PokemonService(pokemon_pb2_grpc.PokemonServiceServicer):

    def Get(self, request, context):
        # TODO: write the server logic here
        return pokemon_pb2.Pokemon()


def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    pokemon_pb2_grpc.add_PokemonServiceServicer_to_server(PokemonService(), server)
    server.add_insecure_port('[::]:50051')
    server.start()
    try:
        while True:
            time.sleep(60 * 60 * 24)
    except KeyboardInterrupt:
        server.stop(0)


if __name__ == '__main__':
    serve()
```

Write the server

```
# imports are omitted for clarity

class PokemonService(pokemon_pb2_grpc.PokemonServiceServicer):   -> define a class that subclasses the generated
                                                                    class
    def Get(self, request, context):
        # TODO: write the server logic here
        return pokemon_pb2.Pokemon()
```

```
def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    pokemon_pb2_grpc.add_PokemonServiceServicer_to_server(PokemonService(), server)
    server.add_insecure_port('[::]:50051')
    server.start()
    try:
        while True:
            time.sleep(60 * 60 * 24)
    except KeyboardInterrupt:
        server.stop(0)


if __name__ == '__main__':
    serve()
```

Write the server

```
# imports are omitted for clarity

class PokemonService(pokemon_pb2_grpc.PokemonServiceServicer):

    def Get(self, request, context):
        # TODO: write the server logic here
        return pokemon_pb2.Pokemon()
```

-> this class should also implements all of the PokemonService service methods

```
def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    pokemon_pb2_grpc.add_PokemonServiceServicer_to_server(PokemonService(), server)
    server.add_insecure_port('[::]:50051')
    server.start()
    try:
        while True:
            time.sleep(60 * 60 * 24)
    except KeyboardInterrupt:
        server.stop(0)


if __name__ == '__main__':
    serve()
```

Write the server

```
# imports are omitted for clarity

class PokemonService(pokemon_pb2_grpc.PokemonServiceServicer):

    def Get(self, request, context):
        # TODO: write the server logic here
        return pokemon_pb2.Pokemon()


def serve():                                                    -> define the entry point for the server
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    pokemon_pb2_grpc.add_PokemonServiceServicer_to_server(PokemonService(), server)
    server.add_insecure_port('[::]:50051')
    server.start()
    try:
        while True:
            time.sleep(60 * 60 * 24)
    except KeyboardInterrupt:
        server.stop(0)


if __name__ == '__main__':
    serve()
```

Write the server

```
# imports are omitted for clarity

class PokemonService(pokemon_pb2_grpc.PokemonServiceServicer):

    def Get(self, request, context):
        # TODO: write the server logic here
        return pokemon_pb2.Pokemon()
```

```
def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))                    -> initialize server
    pokemon_pb2_grpc.add_PokemonServiceServicer_to_server(PokemonService(), server)
    server.add_insecure_port(‘[::]:50051')
    server.start()
    try:
        while True:
            time.sleep(60 * 60 * 24)
    except KeyboardInterrupt:
        server.stop(0)


if __name__ == '__main__':
    serve()
```

Write the server

```
# imports are omitted for clarity

class PokemonService(pokemon_pb2_grpc.PokemonServiceServicer):

    def Get(self, request, context):
        # TODO: write the server logic here
        return pokemon_pb2.Pokemon()
```

```
def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    pokemon_pb2_grpc.add_PokemonServiceServicer_to_server(PokemonService(), server)      -> register the server
    server.add_insecure_port('[::]:50051')
    server.start()
    try:
        while True:
            time.sleep(60 * 60 * 24)
    except KeyboardInterrupt:
        server.stop(0)


if __name__ == '__main__':
    serve()
```

Write the server

```
# imports are omitted for clarity

class PokemonService(pokemon_pb2_grpc.PokemonServiceServicer):

    def Get(self, request, context):
        # TODO: write the server logic here
        return pokemon_pb2.Pokemon()


def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    pokemon_pb2_grpc.add_PokemonServiceServicer_to_server(PokemonService(), server)
    server.add_insecure_port(‘[::]:50051')                                    -> add port
    server.start()
    try:
        while True:
            time.sleep(60 * 60 * 24)
    except KeyboardInterrupt:
        server.stop(0)


if __name__ == '__main__':
    serve()
```

Write the server

```
# imports are omitted for clarity

class PokemonService(pokemon_pb2_grpc.PokemonServiceServicer):

    def Get(self, request, context):
        # TODO: write the server logic here
        return pokemon_pb2.Pokemon()
```

```
def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    pokemon_pb2_grpc.add_PokemonServiceServicer_to_server(PokemonService(), server)
    server.add_insecure_port(‘[::]:50051')
    server.start()                                                    -> start the server
    try:
        while True:
            time.sleep(60 * 60 * 24)
    except KeyboardInterrupt:
        server.stop(0)


if __name__ == '__main__':
    serve()
```

Write the server

```
# imports are omitted for clarity

class PokemonService(pokemon_pb2_grpc.PokemonServiceServicer):

    def Get(self, request, context):
        # TODO: write the server logic here
        return pokemon_pb2.Pokemon()


def serve():
    server = grpc.server(futures.ThreadPoolExecutor(max_workers=10))
    pokemon_pb2_grpc.add_PokemonServiceServicer_to_server(PokemonService(), server)
    server.add_insecure_port(‘[::]:50051')
    server.start()
    try:                                                    -> keep the server alive unless there is a
        while True:                                            KeyboardInterrupt signal
            time.sleep(60 * 60 * 24)
    except KeyboardInterrupt:
        server.stop(0)


if __name__ == '__main__':
    serve()
```

Write the client

```
#!/usr/bin/env python3

from __future__ import print_function

import grpc

import pokemon_pb2
import pokemon_pb2_grpc

POKEMON_ID =  # fill this in


def run():
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = pokemon_pb2_grpc.PokemonServiceStub(channel)
        response = stub.Get(pokemon_pb2.GetReq(id=POKEMON_ID))
    print("Client received: Pokemon #%d is %s" % (response.id, response.name))


if __name__ == '__main__':
    run()
```

Write the client

```
#!/usr/bin/env python3

from __future__ import print_function

import grpc

import pokemon_pb2
import pokemon_pb2_grpc

POKEMON_ID =  # fill this in


def run():                                                    -> define the entry point of the client
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = pokemon_pb2_grpc.PokemonServiceStub(channel)
        response = stub.Get(pokemon_pb2.GetReq(id=POKEMON_ID))
    print("Client received: Pokemon #%d is %s" % (response.id, response.name))


if __name__ == '__main__':
    run()
```

Write the client

```
#!/usr/bin/env python3

from __future__ import print_function

import grpc

import pokemon_pb2
import pokemon_pb2_grpc

POKEMON_ID =  # fill this in


def run():
    with grpc.insecure_channel('localhost:50051') as channel:        -> initialize the grpc client
        stub = pokemon_pb2_grpc.PokemonServiceStub(channel)
        response = stub.Get(pokemon_pb2.GetReq(id=POKEMON_ID))
    print("Client received: Pokemon #%d is %s" % (response.id, response.name))


if __name__ == '__main__':
    run()
```

Write the client

```
#!/usr/bin/env python3

from __future__ import print_function

import grpc

import pokemon_pb2
import pokemon_pb2_grpc

POKEMON_ID =  # fill this in


def run():
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = pokemon_pb2_grpc.PokemonServiceStub(channel)          -> set up client stub for the interaction
        response = stub.Get(pokemon_pb2.GetReq(id=POKEMON_ID))
    print("Client received: Pokemon #%d is %s" % (response.id, response.name))


if __name__ == '__main__':
    run()
```

Write the client

```
#!/usr/bin/env python3

from __future__ import print_function

import grpc

import pokemon_pb2
import pokemon_pb2_grpc

POKEMON_ID =  # fill this in


def run():
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = pokemon_pb2_grpc.PokemonServiceStub(channel)
        response = stub.Get(pokemon_pb2.GetReq(id=POKEMON_ID))       -> do stuff with the stub methods
    print("Client received: Pokemon #%d is %s" % (response.id, response.name))


if __name__ == '__main__':
    run()
```

Write the client

```
#!/usr/bin/env python3

from __future__ import print_function

import grpc

import pokemon_pb2
import pokemon_pb2_grpc

POKEMON_ID =  # fill this in


def run():
    with grpc.insecure_channel('localhost:50051') as channel:
        stub = pokemon_pb2_grpc.PokemonServiceStub(channel)
        response = stub.Get(pokemon_pb2.GetReq(id=POKEMON_ID))
    print("Client received: Pokemon #%d is %s" % (response.id, response.name))    -> print the data out 😁


if __name__ == '__main__':
    run()
```

Verify that the gRPC works

- Start the server
  - `python3 server.py`
- In another window, start the client
  - `python3 client.py`
- Verify that the client received the response

another quick demo

Who are using GRPC?

- **Square** — replacement for all of their internal RPC. One of the very first adopters and contributors to gRPC.
- **CoreOS** — Production API for etcd v3 is all in gRPC
- **Google** — Production API for Google Cloud Services (such as PubSub, ML services) are in gRPC
- **Netflix**, and much more

Should I move to gRPC?

- **Yes**
  - **Binary protocol:** fast, no more text parsing
  - **Native stream:** no need for websockets, streaming is possible
  - **Client Libraries:** we get this for free, gratis!
    - **Stream Multiplexing, Header Compression**
- **but,**
  - **Breaking API changes:** still developing
  - **Poor documentation for some languages**
  - **Load balancing, Error handling**
  - **No standardization across languages, yet**

questions?

terima kasih